INRIA

UNITÉ DE RECHERCHE
INRIA-ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél. (1) 39 63 55 11

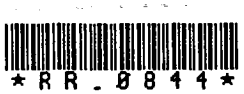
Rapports de Recherche

N° 844

AN ITERATIVE EUCLIDEAN ALGORITHM

Paul CAMION

MAI 1988



AN ITERATIVE EUCLIDEAN ALGORITHM

Paul Camion

INRIA
Domaine de Voluceau, Rocquencourt, B.P. 105
78153 LE CHESNAY Cedex - FRANCE

ABSTRACT

On the basis of the results by JL. DORNSTETTER [3] showing the equivalence between BERLEKAMP's and EUCLID's algorithm, we present an iterative Euclidean extended algorithm. We show how all polynomials obtained by the classical extended Euclidean algorithm are actually automatically produced by that iterative process.

In sum, an algorithm is given which is as economical as BERLEKAMP's for decoding and which is proved to perform decoding of alternant codes by the simple argument used for the EUCLIDEAN algorithm.

Finally a result of BERLEKAMP's [1] is exploited to reduce by another half the degrees of all polynomials involved in the decoding process in the particular case of binary BCH codes.

UN ALGORITHME D'EUCLIDE ITERATIF

RESUME

En nous fondant sur les résultats de JL. DORNSTETTER [3] qui démontrent l'équivalence de l'algorithme de BERLEKAMP et de celui d'EUCLIDE, nous présentons un algorithme d'EUCLIDE étendu itératif. Nous montrons comment tous les polynômes obtenus dans l'algorithme d'EUCLIDE étendu classique sont produits automatiquement dans le processus itératif.

En résumé, nous donnons un algorithme qui est aussi économique que celui de BERLEKAMP pour le décodage et dont la justification pour le décodage des codes alternants est celle de l'algorithme d'EUCLIDE utilisé à cet effet.

Finalement, un résultat de BERLEKAMP est exploité de façon à diviser une nouvelle fois par deux les degrés des polynomes utilisés dans le processus de décodage dans le cas particulier des codes BCH binaires.

Keywords : iterative, algorithm, Euclidean, decoding, alternant code.

1 INTRODUCTION

In a recent paper [3], JL. DORNSTETTER showed how BERLEKAMP-MASSEY iterative algorithm (shortly B-M iterative algorithm in the following) for decoding alternant codes can be derived from a normalized version of the Euclidean extended algorithm. He shows how polynomials $\sigma_i(x)$ and $\omega_i(x)$ processed in the B-M iterative algorithm are the reciprocals of polynomials $U_i(x)$ and $V_i(x)$ occuring in the Euclidean algorithm with the general relation

$$(1) \quad U_i R_0 - V_i R_{-1} = (-1)^i R_i,$$

for two given polynomials R_{-1} and R_0, with deg R_0 < deg R_{-1} and where the R_i, U_i, V_i are related by the Euclidean division :

$$(2) \quad R_{i-1} = Q_i R_i + R_{i+1} \ , \ \deg R_{i+1} < \deg R_i$$

$$(3) \quad U_{i+1} = Q_i U_i + U_{i-1} \ , \ V_{i+1} = Q_i V_i + V_{i-1},$$

with the initial values

$$(4) \quad U_{-1} = V_0 = 0 \ , \ U_0 = V_{-1} = 1.$$

JL. DORNSTETTER shows that for d = deg U_i :

$$(5) \quad \sigma_i(X) = X^d U_i(X^{-1}) \ , \ \omega_i(X) = X^{d-1} V_i(X^{-1}).$$

He shows hows the B-M iterative algorithm avoids constructing explicitly polynomials R_i, i=1, 2, ... The needed coefficients of polynomials Q_i, i = 0, 1, ... are obtained while processing $\sigma_i(X)$, $\omega_i(X)$, i=1, 2, ... together with intermediate polynomials of which the degrees increase from zero to the final degree. This is why the Euclidean algorithm needs roughly twice as many operations as the B-M iterative algorithm in the case of decoding. We here transform the BERLEKAMP-MASSEY iterative algorithm as presented by JL. DORNSTETTER into successive versions of an iterative Euclidean algorithm.

Since we know that the B-M iterative algorithm operates reciprocal polynomials, what is shifted to the left should be here shifted to the right in a corresponding iterative Euclidean algorithm. But when two polynomials are involved, shifting one of them to the right before a linear combination gives essentially the same result as shifting the other to the left. That is what is done in the first version of the presented iterative Euclidean algorithm in which the sequence $R_1, R_2, \ldots, R_i, \ldots$ of remainders are obtained, each one beeing

shifted to the left by a well known d_{j_i} number of positions. Here the logical tests are those introduced by JL. DORNSTETTER. They are here used to operate the sequence of divisions (2) in an iterative process. Sections 3 and 5 give a detailed proof of the process.

Next we transform that iterative Euclidean algorithm in order not to construct the remainders R_i but only the needed polynomials U_i, i=1, ... of "small degrees". We then introduce a last modification in the process to obtain the normalized Euclidean algorithm which as shown by JL. DORNSTETTER is the one that exactly corresponds, elementary operation to elementary operation, to the B-M iterative algorithm.

The result is that we obtain an iterative process in which every operation is understood as an elementary step in the sequence (2) of divisions. That process is automatical with logical tests on the number j of iterations already undergone and on the nullity of the content of a given fixed register. Moreover it is as economical as the B-M iterative algorithm.

2 INTRODUCING ITERATIVE STEPS IN EUCLID'S ALGORITHM

2.1 The usual Euclidean Algorithm

We run the series of divisions

$$(1) \quad R_{i-1} = Q_i R_i + R_{i+1}, \quad \deg R_{i+1} < \deg R_i,$$

for i = 0, ... up to the value given by a **stop test**. That value may be the smallest k such that $R_{k+1} = 0$, that is the case when we compute $\gcd(R_{-1}, R_0) = R_k$. But it also can be the value k such that $2\deg R_{k-1} \geq \deg R_{-1} = n$ and $2\deg R_k < n-2$. This is the case in the decoding algorithm, where $R_{-1} = X^n$ and R_0 is the syndrom S(X). (MC WILLIAMS and SLOANE [4] pages 362-367).

2.2 Shifts

For a polynomial of degree m, we write

$$a_m X^m + a_{m-1} X^{m-1} + \ldots + a_0.$$

Thus mutliplying by X a polynomial will be considered a **shift to the left.**

2.3 The operations

The considered polynomials are in K[X] for any field K. For polynomials A and B we will repeatedly perform operations of the following kind :

i) $B \leftarrow XB$
ii) $B \leftarrow \alpha B$, for some α in K
iii) $A \leftarrow A - B$.

2.4 The degrees of the operands

Let n be the degree of R_{-1}.

All the way to the end, the operands are possibly shifted to the left. They are actually multiplied by a monomial, say X^s in order that whenever operation iii) is performed, then deg A = deg B = n.

This is because **we avoid to check the degree of any polynomial.** That would force us to verify which is the non-zero coefficient of highest degree. But in the course of the division of R_{i-1} by R_i, we know that we get the remainder R_{i+1} by the condition : deg R_{i+1} < deg R_i . We then know that we have to make

$$R_{i-1} \leftarrow R_i \ , \quad R_i \leftarrow R_{i+1},$$

and then to go on until the stop test on i is met. Since we don't have a criterium based upon degrees, we will consider an integer parameter d_j of which the relation with degrees of the concerned polynomials is clarified in Property 4.1. Moreover index j of d_j counts up the numbers of steps where operation i) was performed, thus j counts up the number of shifts. It also counts up the number of iterations.

3 THE ITERATIVE EUCLIDEAN ALGORITHM, DESCRIPTION AND PROOF

3.1 Bounding the degrees of the polynomials

We here refer to the process given in Appendix 1.

When entering 3b, E'_j is the dividend polynomial and E_j is the divisor polynomial. But there, they are interchanged and in the following steps through

3c, E'_j is the fixed divisor and E_j takes the successive values of intermediate remainders.

Now there, E'_j is not exactly one of the successive divisors R_i in 2(1), but such a divisor multiplied by a monomial in order **that deg E'_j always equals n**, the degree of R_{-1}.

The degree of E'_j certainly is n for j=0 after initialization. Further, E'_j only changes when passing through 3b. There it takes the value of E_j which has degree n by the condition of entering 3b. It is assumed that in the initial situation,
$\deg R_0 < \deg R_{-1} = n$.

At steps 2 and 3a, we see that if $\deg E_j < n$, then E_j is shifted to the left. When a substraction is performed in 3b or 3c, then $\deg E_j = \deg E'_j = n$ and for the resulting E_{j+1}, we have that $\deg E_{j+1} < n$ since for that substraction, we always have that $\Delta = [X^n] E'_j$, $\Delta_j = [X^n] E_j$, where we denote by $[X^\ell] P$ the coefficient of X^ℓ in polynomial P.

Since $\deg E_0 \leq n$, **we then always have that $\deg E_j \leq n$.**

To sum up, we state

Property 3.2

Entering any step of the algorithm, polynomial E'_j has degree n. Regarding polynomial E_j, its degree is never larger than n and equals n if the discrepancy Δ_j does not cancel. Entering 3b or 3c, polynomials E'_j and E_j both have degree n and the linear combination there produces a polynomial with degree less than n.

3.3 Introducing some notations

Let $j_{-1} < j_0 < \ldots < j_i$ denote the successive steps where conditions to enter 3b are fulfilled. That sequence is actually denoted by (j_{i-1}) since integer i takes the values 0, 1, 2, ... **Notice that d_j only changes when passing through 3b.** We thus have that

(1) $$d_j = d_{j_i} \quad \text{for} \quad j = j_{i-1} + 1, \ldots, j_i.$$

At step $j = j_{i-1}$ then d_j changes from the value $d_{j_{i-1}}$ to the value $d_{j_i} = j+1 - d_{j_{i-1}}$, and since $j \geq 2d_j$, then

$$2d_{j_i} = j + (j+2 - 2d_j) > j+1.$$

Hence there is a step t_i after j_{i-1} for which

(2) $$t_i = 2d_{t_i} = 2d_{j_i}$$

Instants t_{i-1} and t_i together with instant j_{i-1} define two phases in the process of the algorithm.

phase b_i

Phase b_i is the succession of steps starting at t_{i-1} and ending when step j_{i-1} is completed.

During phase b_i, we enter repeatedly 3a until passing through 3b at step j_{i-1}.

Phase c_i

Exiting phase b_i, starts a sequence of steps through 3a or 3c up to the instant $t_i - 1$.

Property 3.4

The number of steps in phase b_i equals that one in phase c_i and is worth $d_{j_i} - d_{j_{i-1}}$.

Those numbers are respectively $j_{i-1} - t_{i-1} + 1$ and $t_i - 1 - j_{i-1}$.

By definition of d_{j+1}, in 3b, we have that $j_{i-1} = d_{j_{i-1}} + d_{j_i} - 1$.

The property then follows by (2).

Since after initialization we start with phase b_0, we set $t_{-1} = 0$.

Remark

If when entering phase b_i, we have that $j = t_{i-1} = 2d_{j_{i-1}}$ and $\Delta j \neq 0$, we will enter 3b right away and the number of steps in b_i as in c_i is 1. This is the case where the quotient Q_i has degree 1 and they are only two substractions in

the process of division. In that case, step 3a is not used. We will see that more generally deg $Q_i = d_{j_i} - d_{j_{i-1}}$ and this is measured up by the number of steps in phase b_i.

3.5 The first iterations of the algorithm

Since deg $R_0 <$ deg $R_{-1} = n$, then Δ_0 is possibly 0. If so, we then enter a loop : 2, 3a, 4, 2, ... up to the moment where deg E_j is n. Since d_j is still worth zero, we enter 3b. Refering to the beginning of section 3.3, we there have $j = j_{-1}$ and as observed in 3.3, we will have that $d_{j+1} = d_{j_0}$.

We have that

$$d_{j+1} = j+1 - d_j = j+1 = n - \deg R_0.$$

We may write as well

(3) $$d_{j_0} - d_{j_{-1}} = \deg Q_0.$$

Next statement asserts that this situation continues up to the end.

4. THE VALUES OF THE OBTAINED POLYNOMIALS AT CRUCIAL STEPS

We need to be able to get every polynomial of the sequence R_i, i = 0,... of 2 (1) from the obtained values of E'_j and E_j. The crucials steps are the steps j_i and t_i defined in 3.3, i=0, ...

Property 4.1

Let j_i and t_i be the steps defined in 3.3, for i = 0, ... We have that

(1) $$E'_{t_i} = X^{d_{j_i}} R_i \ , \ E_{t_i} = X^{d_{j_i+1}} R_{i+1}, \ i = 0, \ \ldots$$

Moreover

(2) $$d_{j_i} = n - \deg R_i$$

From (2) we also will have that

(3) $$d_{j_{i+1}} - d_{j_i} = \deg R_i - \deg R_{i+1}, \quad i=0, \ldots$$

Since we also need (3) for the proof by recurrence on i, we first verify (1), (2) and (3) for i = -1. Notice that, for logical correctness, we should have written i-1 in place of i in (1) and (2) since sequence (i) actually starts from zero. That wouldn't have been aesthetic.

Remember that $t_{-1} = d_{j_{-1}} = 0$. Then, for i=-1, (1) is given by the initialization as well as (2). Relation (3) for i=-1 is precisely the relation stated as 3(3).

We now assume that all relations are verified where entering step t_{i-1} and we prove they are valid when entering step t_i. At step t_{i-1} we enter phase b_i and when entering the last step of phase b_i, then $E_{t_{i-1}}$ underwent a number of shifts equal to $d_{j_i} - d_{j_{i-1}} - 1$, by Property 3.4. Since we only went thru 3a, then $E_{t_{i-1}}$ is then shifted into

$$X^{d_{j_i} - d_{j_{i-1}} - 1 + d_{j_{i-1}+1}} R_i .$$

At the conclusion of 3b, at step $j = j_{i-1} + 1$, this becomes the value of E'_j and then E'_j remains unchanged during phase c_i, that is, up to $j = t_i$.

We now verify the second relation of (1). Here we start with $E'_{t_{i-1}}$ which remains unchanged in 3a all the time of phase b_i except for the last step which is step j_{i-1}. There, E_j takes the value of E'_j which is the one of $E'_{t_{i-1}} = X^{d_{j_{i-1}}} R_{i-1}$. We then enter phase c_i. The operation in the end of 3b plus the operations in phase c_i are those of a regular division of R_{i-1} by R_i except that here the divisor is never shifted and instead, the intermediate remainder is shifted once to the left after it is obtained by a substraction. The number of such operations is

$$d_{j_i} - d_{j_{i-1}} + 1,$$

by Property 3.4.

By the recurrence hypothesis, that number is $\deg R_{i-1} - \deg R_i + 1$, which is $\deg Q_i + 1$. But this is exactly the number of substractions in the usual division. This means that we obtain at the end of phase c_i a polynomial which is R_{i+1} shifted a certain number of times which is the number of steps in phase c_i plus the last step in phase b_i, that is $d_{j_i} - d_{j_{i-1}} + 1$ plus the possibly large shift that we

started with. Since we started with the dividend $X^{d_{j_{i-1}}} R_{i-1}$, that shift is made of $d_{j_{i-1}}$ elementary shifts. Hence

$$E_{t_i} = X^{d_{j_i+1}} R_{i+1}.$$

To end the proof, we have to show (2) and (3).

By the first assertion of Property 3.2, we have that deg E'_{t_i} = n. This implies (2).

We also know by (1) that

$$\deg E'_{t_i} - \deg E_{t_i} = \deg R_i - \deg R_{i+1} - 1.$$

But that number is the necessary number of shifts, starting at step t_i when entering b_{i+1} to satisfy the condition of entering 3b, which is $\Delta_j \neq 0$, or in other words deg E_j = n (remember that we have $2d_j \leq j$ in phase b_{i+1} thoroughly).

By Property 3.4, that number is $d_{j_{i+1}} - d_{j_i} - 1$, and this ends the proof.

We now examine the successive values of E_j and E'_j thru 3b and 3c.

Property 4.2

For $0 < k < \deg Q_i$, we have that

$$(4) \quad E_{j_{i-1}+k} = X^{d_{j_{i-1}}+k}(R_{i-1} - Q_i^{(k)} R_i),$$

$$(5) \quad E'_{j_{i-1}+k} = X^{d_{j_i}} R_i,$$

where

$$(6) \quad Q_i^{(k)} = q_e X^e + \ldots + q_{e-k+1} X^{e-k+1} = Q_i - q_{e-k} X^{e-k} - \ldots - q_0.$$

Notice that $d_{j_{i-1}} + e = d_{j_i}$, by Property 4.1.

By Property 4.1, we have that $E'_{t_{i-1}} = X^{d_{j_{i-1}}} R_{i-1}$ and when entering 3b at step j_{i-1} we have that $E'_{j_{i-1}} = E'_{t_{i-1}}$

Moreover, by Properties 3.2 and 4.1, we have that

(7) $E_{j_{i-1}} = X^{d_{j_i} - d_{j_{i-1}} - 1} E_{t_{i-1}} = X^{d_{j_i}} R_i$.

Remember now that E'_j and E_j will be interchanged.

Thus passing thru 3b, we have that

(8) $E_{j_{i-1}+1} = X(X^{d_{j_{i-1}}} R_{i-1} - X^{d_{j_{i-1}}} Q_i^{(1)} R_i)$.

Now for $k \geq 2$ we assume that

(9) $E_{j_{i-1}+k-1} = X(X^{d_{j_{i-1}}+k-2} R_{i-1} - X^{d_{j_{i-1}}+k-2} Q_i^{(k-1)} R_i)$.

Then passing thru 2, we have that

(10) $\Delta_{j_{i-1}+k-1} = [X^n] \; E_{j_{i-1}+k-1}$

and by Property 3.2 :

(11) $q_{e-k+1} = \Delta_{j_{i-1}+k-1} \; \Delta^{-1}$,

where $\Delta = [X^n] \; E_{j_{i-1}} = [X^n] \; X^{d_{j_i}} R_i$, by (7).
Whence

(12) $E_{j_{i-1}+k} = X(E_{j_{i-1}+k-1} - q_{e-k+1} \; X^{d_{j_i}} R_i)$
$= X(E_{j_{i-1}+k-1} - q_{e-k+1} \; X^{d_{j_{i-1}+k-1}} \; X^{e-k+1} R_i)$

Finally,

(13) $E_{j_{i-1}+k} = X(X^{d_{j_{i-1}}+k-1} R_{i-1} - X^{d_{j_{i-1}}+k-1} Q_i^{(k)} R_i)$.

Remark 4.3

We see that for $k = e+1$, then $Q_i^{(k)}$ equals Q_i. Also $d_{j_{i-1}+e} = d_{j_i}$, by Property 4.1. We also have that $j_{i-1}+e+1 = j_{i-1}+1+ d_{j_i} - d_{j_{i-1}}$, which by Property 3.4 is t_i. Thus for $k = e+1$, that is when the division process of R_{i-1} by R_i ends, then we have that

(14) $E_{t_i} = X^{d_{ji}+1} R_{i+1}$,

as already obtained in Property 4.1.

5 THE STOP TEST

5.1 Computing a gcd

In the case where we compute gcd (R_{-1}, R_0), we eventually have deg $R_i = 0$ and then

$$R_i = 0 \Rightarrow \gcd(R_{-1}, R_0) = R_{i-1}$$
$$R_i \neq 0 \Rightarrow \gcd(R_{-1}, R_0) = 1.$$

Thus when for the first time, a newly computed $d_{j+1} = d_{j_i}$ in 3b is worth n, then we have that $E'_{j+1} = E'_{t_i} = X^n R_i$ which is either 0 or a non-zero scalar times X^n. This is a consequence of Property 4.1.

5.2 Decoding alternant codes

In the case where the Euclidean algorithm is used for decoding pure errors in an alternant code with minimum distance $d \geq r+1$, where r is even, then $R_{-1} = X^r$ and we know that the error evaluator polynomial is a scalar multiple of R_k where k is the smallest integer for which

$$\deg R_{k-1} \geq r/2, \deg R_k \leq r/2 - 1$$

([4] page 366).

We may by 4(2) state that condition in words of d_{j_k} :

$$d_{j_{k-1}} \leq r/2, \quad d_{j_k} \geq r/2 + 1.$$

6. AN ITERATIVE VERSION OF THE EUCLIDEAN ALGORITHM WHERE THE REMAINDERS ARE NOT ACTUALLY COMPUTED.

6.1 Why computing the remainders could and should be avoided.

The degree of R_{-1} is here denoted by r.

Since we actually need in the case of correcting codes the values of U_k and R_k, then it is worthwhile to avoid computing the sequence of remainders $R_1, R_2, \ldots, R_{k-1}$ which in the worst case needs computing r-1 coefficients, then

r-2 coefficients, ..., and finally k+1 coefficients. Notice that when the quotient Q_j has degree 1, then computing the remainder R_{j+1} of degree ℓ needs passing thru 3b and 3c once and there processing $2(\ell+2)$ products and $2(\ell+1)$ substractions.

Let us remind that

$$\deg U_i = \deg R_{-1} - \deg R_{i-1}.$$

That is easily verified by recurrence on i.

Observe that by 4(2), we have that $d_{j_{i-1}} = \deg U_i$.
Then if we compute the sequence of polynomials $U_1,\ldots,U_k$ instead, since

$$\deg U_k = r - \deg R_{k-1} \leq r/2,$$

then all polynomials processed have small degrees.

The basic observation in JL. DORNSTETTER [3] is that sequence (U_i) and (V_i) may be computed without computing sequence (R_i). All we need to know are the initial values R_{-1} and R_0. We have that $U_{-1} = V_0 = 0$, $U_0 = V_{-1} = 1$. This means that we shall be able to compute sequence (Q_i) which gives

(1) $$U_{i+1} = Q_i U_i + U_{i-1}\ ,\ V_{i+1} = Q_i V_i + V_{i-1}\ ,$$

whithout performing the sequence of divisions :

(2) $$R_{i-1} = Q_i R_i + R_{i+1},\ \deg R_{i+1} < \deg R_i\ ,$$

which would produce at step i, Q_i and R_{i+1}.

6.2 How to get the coefficients of Q_i whithout computing R_{i+1}

This relies on the general relation

(3) $$U_i R_0 - V_i R_{-1} = (-1)^i R_i.$$

We immediatly show how the coefficients of polynomial Q_i, which successively appear while performing the division of R_{i-1} by R_i, may be computed when knowing U_{i-1}, U_i, V_{i-1} and V_i.

For, let

$$(4) \qquad Q_i^{(k)} = q_e X^e + \ldots + q_{e-k+1} X^{e-k+1}$$

be the polynomial whose terms are the first k terms with higher degrees form Q_i. Denote by r_i the degree of R_i and let Γ be the leading coefficient of R_i, thus

$$(5) \qquad \Gamma = [X^{r_i}] R_i .$$

Now we use the notation

$$(6) \qquad \Gamma_{k-1} = [X^{r_{i-1}-k+1}] (R_{i-1} - Q_i^{(k-1)} R_i).$$

This means that after performing k-1 substractions, $k \geq 2$, in the process of dividing R_{i-1} by R_i we got the polynomial $R_{i-1} - Q_i^{(k-1)} R_i$ with degree at most $r_{i-1}-k+1$. Next step in that division is the performing of

$$(7) \qquad R_{i-1} - Q_i^{(k-1)} R_i - \Gamma_{k-1} \Gamma^{-1} X^{e-k+1} R_i$$

which produces a polynomial of degree at most $r_{i-1} - k$. If $k=e+1$, then the division is completed and the obtained polynomial is R_{i+1} with $\deg R_{i+1} = r_{i+1} < r_i = r_{i-1} - e$. Since (6) only defines Γ_{k-1} for $k>2$, that is, after at least one substraction was performed, we need defining Γ_0 which necessarily is worth $[X^{r_{i-1}}] R_{i-1}$ since $q_e = \Gamma_0 \Gamma^{-1}$, with Γ defined by (5). But subsequently, we compute at k^{th} step

$$(8) \qquad q_{e-k+1} = \Gamma_{k-1} \Gamma^{-1}.$$

Now relation (3) allows computing Γ when knowing U_i and V_i but without actually computing R_i. We indeed have that

$$(9) \qquad [X^{r_i}] R_i = (-1)^i [X^{r_i}] (U_i R_0 - V_i R_{-1})$$

$$= (-1)^i [X^{r_i}] U_i R_0 - (-1)^i [X^{r_i}] V_i R_{-1}$$

This means that **we only have to compute one coefficient** in each polynomial product $U_i R_0$ and $V_i R_{-1}$. Now we need computing the sequence Γ_{k-1} in order to get the k^{th} coefficient q_{e-k+1}, k=1, 2, ... Since Γ_0 is the previous value of Γ, we then compute Γ_{k-1} for $k > 1$ by the relation

(11) $(U_{i-1}+Q_i^{(k-1)}U_i)R_0 - (V_{i-1}+Q_i^{(k-1)}V_i)R_{-1}$

$=(-1)^{i-1}(R_{i-1}-Q_i^{(k-1)}R_i)$,

and from there

(12) $Q_i^{(k)} = Q_i^{(k-1)} + \Gamma_{k-1}\,\Gamma^{-1}\,X^{e-k+1}$

Remark 6.3

If R_{-1} is reduced to X^r, then

$$\Gamma_{k-1} = (-1)^{i-1}[X^{r_{i-1}-k+1}]\,(U_{i-1} + Q_i^{(k-1)}\,U_i)\,R_0.$$

This is true by (11) for every i since k is larger than 1 and

$$r_{i-1} = \deg R_{i-1} \leq n.$$

6.4 The polynomials G_j.

In Appendix 2 we replaced all E_j's by G_j's and H_j's. But whenever $E_j-\Delta_j\Delta^{-1}E'_j$ appears, then a plus is substituted to the minus to get $G_j+\Delta_j\Delta^{-1}G'_j$ or $H_j+\Delta_j\Delta^{-1}H'_j$. The initial conditions are $G_0 = X\,U_0$, $G'_0 = U_{-1}$, $H_0 = X\,V_0$, $H'_0 = V_{-1}$, where $U_{-1} = V_0 = 0$, $V_{-1} = U_0 = 1$. With the same notations as for Property 4.2, we have

Property 6.4.1

With the notations of Property 4.1, we have that

(1) $G_{j_{i-1}} = G'_{t_i} = X^{d_{j_i}}\,U_i, G_{t_i} = X^{d_{j_i+1}}\,U_{i+1}, i=0,\ldots$

$H_{j_{i-1}} = H'_{t_i} = X^{d_{j_i}}\,V_i, H_{t_i} = X^{d_{j_i+1}}\,V_{i+1}, i=0,\ldots$

We leave the proof to the reader.

Property 6.4.2

For $0 < k < \deg Q_i$, we have that

$G_{j_{i-1}+k} = X^{d_{j_{i-1}+k}}\,(U_{i-1}+Q_i^{(k)}U_i)$, $G'_{j_{i-1}+k} = X^{d_{j_i}}\,U_i$;

$H_{j_{i-1}+k} = X^{d_{j_{i-1}+k}}\,(V_{i-1}+Q_i^{(k)}V_i)$, $H'_{j_{i-1}+k} = X^{d_{j_i}}\,V_i$.

The proof is the same as the one for Property 4.2. except that $\Delta_{j_{i-1}+k-1}$ is here obtained as

$$\Delta_{j_{i-1}+k-1} = (-1)^{i-1}[X^n]\,(G_{j_{i-1}+k-1}\,R_0 - H_{j_{i-1}+k-1}\,R_{-1})$$

and Δ as

$$\Delta = (-1)^{i-1}[X^n]\,(G_{j_{i-1}}\,R_0 - H_{j_{i-1}}\,R_{-1})$$

6.5 The stop test

We see in the data of the extended iterative Euclid algorithm given in Appendix 2 that the successive values of Δ_j and of d_j are the same as those from the iterative algorithm of Appendix 1. Then the stop test is the same as in section 5.

Remark 6.6

When R_{-1} is reduced to X^r, then by Property 6.4.2, $H_{j_{i-1}+k}\,R_{-1}$ has no term of degree less than r+1 since $d_{j_{i-1}+k}$ always is at least one. On the other hand, as for Remark 4.3, we can see that $H_{t_{i-1}} = X^{d_{j_{i-1}+1}}\,V_i$ and since by Property 3.4 we have that $H_{j_{i-1}} = X^{d_{j_i}-d_{j_{i-1}}-1}\,H_{t_{i-1}}$, then $H_{j_{i-1}} = X^{d_{j_i}}\,V_i$. Now for $i \geq 0$, we know that d_{j_i} is larger than zero and consequently $H_{j_{i-1}}\,R_{-1}$ does not bring any contribution in the expression of Δ since its terms with degree r or less all cancel.

To sum up, for $j = j_{i-1}$ up to t_i we don't need the values of H_j to compute Δ_j. Now the steps from t_i up to $j_{i-1}-1$ all take place in 3a. But in 3a, we have that H_j becomes $X\,H_j = H_{j+1}$ which certainly has no term of degree less than n if H_j hasn't.

On the whole we will not need computing the H_j whenever R_{-1} is reduced to X^r.

7. THE NORMALIZED ITERATIVE EXTENDED EUCLIDIEAN ALGORITHM

7.1 Why is a normalized version usefull

We now focus our attention on the iterative version of the extended normalized Euclidean algorithm that exactly corresponds to the B-M algorithm. We here then consider the algorithm for decoding alternant codes. We here

denote by r the degree of R_{-1} which actually is X^r and the stop test is the one given in 5.2. We also use remark 6.6 which spares us the computing of sequence (H_i). Moreover our objective is now to save the time and memory space needed to performing the interchanges in the first lines of 3b.

7.2 Normalization

The division

$$(1) \quad R_{i-1} = Q_i R_i + R_{i+1}$$

that gives Q_i and R_{i+1} is usually performed by starting with the substraction

$$(2) \quad R_{i-1} - b_{i-1} b_i^{-1} X^e R_i$$

where b_s is the leading coefficient or R_s and where e is the degree of Q_i. To save time, the first substraction will be

$$(3) \quad -b_i \; b_{i-1}^{-1} R_{i-1} + X^e R_i$$

we then may write

$$(4) \quad -b_i \; b_{i-1}^{-1} R_{i-1} = - X^e R_i + P_i.$$

where deg $P_i <$ deg R_{i-1}.

What we started doing, together with what will follow, is the sequence of operations in the division of R_{i-1}^* by R_i with

$$(5) \quad R_{i-1}^* = - b_i \; b_{i-1}^{-1} R_{i-1}.$$

We then obtain

$$(6) \quad Q_i^* = -b_i \; b_{i-1}^{-1} Q_i,$$

and by (4), the leading coefficient of Q_i^* is -1.

Starting in that way at i = 0, we obtain sequence (R_i^*) and (Q_i^*) of which the correspondance with (R_i) and (Q_i) is given by

Property 7.3 **The sequence (R_i^*) of remainders produced by the normalized Euclidean algorithm is recursively defined by the division**

$$(7) \quad c_{i-1} R_{i-1}^* = Q_i^* R_i^* + R_{i+1}^* , \deg R_{i+1}^* < \deg R_i^* ,$$

where the scalar c_{i-1} is adjusted in order that the leading coefficient of Q_i^* be -1. Let (R_i) be the sequence of remainders of the classical Euclidean algorithm with the initial datas $R_{-1} = R_{-1}^*$, $R_0 = R_0^*$. We then have

$$(8) \quad R_i^* = a_i R_i ,$$

where a_i is recursively given by

$$(9) \quad a_i b_i + a_{i+1} b_{i-1} = 0 ,$$

b_i, being the leading coefficient of R_i, i = 0, 1, ...

Moreover, sequences (U_i^*) and (V_i^*) for the normalized Euclidean algorithm defined recursively by

$$(10) \quad U_{i+1}^* = Q_i^* U_i^* + c_{i-1} U_{i-1}^* , V_{i+1}^* = Q_i^* V_i + c_{i-1} V_{i-1}^* ,$$

with

$$(11) \quad U_{-1}^* = U_{-1} = V_0^* = V_0 = 0, U_0^* = U_0 = V_{-1}^* = V_{-1} = 1 ,$$

each Q_i^* being given by (7) and where (U_i) and (V_i) are the sequences given by 1(3) in the classical extended algorithm.

We finally have that

(12) $\quad U_i^* = a_i\ U_i,\quad V_i^* = a_i\ V_i,\ i = 0, 1, \ldots$

(13) $\quad Q_i^* = -\ b_{i-1}^{-1}\ b_i\ Q_i,\ \ i = 0, 1, \ldots$

We have that sequence (b_i) is given by the classical sequence R_i ; moreover (7) and (8) define sequences (R_i^*), (Q_i^*),(c_i) and (a_i) while sequence (U_i^*) and (V_i^*) are given by (10) and (11). Now (13) which relates (Q_i^*) and the classical sequence (Q_i) will be proved as well as (9). Next, (12) will be proved by induction on i. By (8), (7) writes

(14) $\quad c_{i-1}\, a_{i-1}\, R_{i-1} = Q_i^*\, a_i\, R_i + R_{i+1}^*,\ \deg R_{i+1}^* < \deg R_i.$

But since

$$c_{i-1}\, a_{i-1}\, R_{i-1} = c_{i-1}\, a_{i-1}\ Q_i\ R_i + c_{i-1}\, a_{i-1}\ R_{i+1},\ \deg R_{i+1} < \deg R_i,$$

we must have that both member cancel in the relation

(15) $\quad (c_{i-1}\, a_{i-1}\ Q_i - a_i\, Q_i^*)\, R_i = R_{i+1}^* - c_{i-1}\, a_{i-1}\, R_{i+1},$

so that coefficient a_{i+1} defined by $R_{i+1}^* = a_{i+1}\, R_i$ must verify, by (15),

(16) $\quad a_{i+1} = c_{i-1}\, a_{i-1}.$

Now by (15), we see that $Q_i^* = v_i\, Q_i$ for some scalar v_i. But since Q_i^* has leading coefficient -1, then (13) is verified. Moreover, by (15) and (13), we have

that

$$(17) \qquad c_{i-1} = - a_{i-1}^{-1} a_i b_{i-1}^{-1} b_i ,$$

We thus see by (16) and (17) that a_{i+1} is recursively given by (9). We now verify (12). The i^{th} relation (10) gives by induction

$$(18) \qquad U_{i+1}^{*} = c_{i-1} a_{i-1} U_{i-1} + Q_i^{*} a_i U_i,$$

and the same for V_{i+1}^{*}.

Using (17), (13), (9) and 1(3) gives actually (12).

For the normalized extended Euclidean algorithm, we have

Property 7.4

$$P_1 : \qquad \deg R_i^{*} < \deg R_{i-1}^{*} = n - \deg U_i^{*}$$

$$P_2 : \qquad U_i^{*} V_{i-1}^{*} - U_{i-1}^{*} V_i^{*} = (-1)^i a_i a_{i-1}$$

$$P_3 : \qquad U_i^{*} R_0 - V_i^{*} R_{-1} = (-1)^i R_i^{*}$$

The proof is by recurrence on i and uses the relations of Property 7.3 Going back thru the preceeding sections with the help of Properties 7.3 and 7.4 allows the justification of the iterative process given by Appendix 3.

8. DECODING BINARY BCH CODES WITH THE ITERATIVE EUCLIDEAN ALGORITHM

8.1 The ring of formal power series IF_2 [[X]]

We here are first concerned with the ring of formal series denoted by A[[X]] where A is any commutative ring with a unity. We refer to N. BOURBAKI [2] for the general properties of that ring. We are here concerned with the case

where A is the finite field $\mathbb{F}_2$ of two elements. We consider any formal series S from $\mathbb{F}_2[[X]]$ without constant term. We write $\tilde{S}$ for the sum of all terms from S with odd degrees. Thus $S = \tilde{S} + \overline{S}$.

We first recall a Theorem of BERLEKAMP [1].

Theorem 8.2

Let S be any formal power series from $\mathbb{F}_2[[X]]$ without constant term. Then the inverse $1+G$ of $1+S$ in $\mathbb{F}_2[[X]]$ is such that $G=\tilde{G}$ iff $S^2 = \overline{S}$.

We here prove a lemma which entails Theorem 8.2 and which leads to an algorithm for constructing the inverse 1+G of 1+S whenever $S^2 = \overline{S}$.

Lemma 8.3

Let S and S_1 be any two series form $\mathbb{F}_2[[X]]$ without constant term. Then for $S = S_1 + S_2$ we have that $S^2 = S_2$ iff $S = \sum_{i\geq 0} S_1^{2^i}$.

The condition is necessary.

Since we have that $S = S_1 + S^2$, equality

$$S = S_1 + S_1^2 + \ldots + S_1^{2^j} + S^{2^{j+1}},$$

assumed by induction, gives

$$S = S_1 + S_1^2 + \ldots + S_1^{2^{j+1}} + S^{2^{j+2}}.$$

Now we have that $S^{2^{j+2}} = X^{2^{j+2}} T$, with T in $\mathbb{F}_2[[X]]$. Thus

$$S = \lim_{i \to \infty} \sum_{j \le i} S_1^{2j} + X^{2j+2} T = \sum_{i \ge 0} S_1^{2^i}$$

The condition is sufficient

Since $S^2 = \sum_{i \ge 1} S_1^{2^i}$, then we have that $S = S_1 + S^2$, hence $S^2 = S_2$.

We particularize the hypothesis of Lemma 8.3 by taking for S_1 the series $\tilde{S}$.

Thus by Lemma 8.3 we have that $S^2 = \bar{S}^2$ iff $S = \sum_{i \ge 0} \tilde{S}^{2i}$.

We now prove Theorem 8.2.

The condition is necessary

We know (see for example [2]) that the inverse 1+S of 1+G writes

(1) $$\sum_{i \ge 0} G^i = \sum_{i \ge 0} G^{2i+1} + \sum_{i \ge 0} G^{2i}.$$

Denoting by $\tilde{S}$ the series $\sum_{i \ge 0} G^{2i+1}$, we have that $S = \tilde{S} + \bar{S}$ with $\bar{S} = S^2$.

The condition is sufficient

By hypothesis we have that

(2) $$S(1+S) = S + S^2 = \tilde{S}$$

But we also have by Lemma 8.3 that

(3) $$S = \tilde{S}(1 + \tilde{S} + \ldots + \tilde{S}^{2^i - 1} + \ldots)$$

Substituting the RHS of (3) for S in the first factor of the LHS of (2) shows that

(4) $$(1+G)(1+S) = 1,$$

with

(5) $$G = \sum_{i \ge 1} \tilde{S}^{2i-1}.$$

Hence $G = \tilde{G}$.

8.4 A fast algorithm to inverse the rational formal power series 1+S whenever $S = \tilde{S} + \bar{S}$ and $\bar{S} = S^2$.

By (3) and (5) we have that

(6) $\tilde{S}G = \bar{S}$.

Thus for

(7) $S = s_1 X + s_2 X^2 + \ldots + s_n X^n + \ldots$

and

(8) $G = g_1 X + g_3 X^3 + \ldots + g_{2n+1} X^{2n+1} + \ldots,$

we have that

(9) $\sum_{0 \le j \le n} g_{2j+1} s_{2(n-j)+1} = s_{2n+2}, n = 0, 1, 2, 3, \ldots .$

Thus if b is the smallest positive integer for which $s_{2b+1} = 1$, we have that

$$
\begin{aligned}
&s_{b+1} = s_{2(b+1)} = g_1 s_{2b+1} \\
&s_{b+2} = s_{2(b+2)} = g_3 s_{2b+1} + g_1 s_{2b+3} \\
(10)\quad &s_{b+3} = s_{2(b+3)} = g_5 s_{2b+1} + g_3 s_{2b+3} + g_1 s_{2b+5} \\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
&s_{b+i} = s_{2(b+i)} = g_{2i-1} s_{2b+1} + g_{2i-3} s_{2b+3} + \ldots + g_1 s_{2(b+i)-1} \\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots
\end{aligned}
$$

from which we compute successively $g_1, g_3, g_5, \ldots, g_{2i-1}, \ldots$.

But since $s_{b+i} = 0$ for $i \le b+1$, we also have that $g_{2j+1} = 0$ for $j \le b$ and we rewrite (10) from i = b+1 up to i= c+1,

$$
\begin{aligned}
&g_{2b+1} = s_{2b+1} = 1 \\
&g_{2b+3} = g_{2b+1} s_{2b+3} + s_{2b+2} \\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
(11)\quad &g_{2(b+j)+1} = g_{2b+1} s_{2(b+j)+1} + g_{2b+3} s_{2(b+j-1)+1} \\
&\qquad + \ldots + g_{2(b+j)-1} s_{2b+3} + s_{2b+j+1} \\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
&g_{2c+1} = g_{2b+1} s_{2c+1} + g_{2b+3} s_{2(c-1)+1} + \ldots + g_{2c-1} s_{2b+3} + s_{b+c+1},
\end{aligned}
$$

where the value of c is to be specified later on.

Remark 8.5

The formal power series 1+S is assumed to be rational, that is, there exists polynomials U an V from $\mathbb{F}_2[X]$ such that V(1+S) = U. We also know that the inverse 1+G of 1+S is the series verifying U(1+G) = V. Our aim is to compute U and V by the iterative extended Euclidean algorithm in the case where S is yielded by the syndrom obtained in decoding a binary BCH code. That syndrom is a polynomial S_0 of degree r-1, where r is even, and the series S is defined by

$$
\begin{aligned}
& V(1+S_0) \equiv U \bmod X^r \\
& \gcd(U,V) = 1, \\
& \deg V \le r/2, \quad \deg U \le r/2-1, \\
& V(1+S) = U.
\end{aligned}
\tag{12}
$$

This is a well known fact about decoding alternant codes ([4] pages 365-369).

Thus we only need the terms of degrees at most r-1 of G and we will solve the problem :

Find polynomials U and V such that

$$
\begin{aligned}
& U(1+G_0) \equiv V \bmod X^r, \\
& \gcd(U,V) = 1, \\
& \deg V \le r/2 \quad , \quad \deg U \le r/2-1,
\end{aligned}
\tag{13}
$$

where G_0 is the sum of the terms with degrees at most r-1 of G. We see that for 2c+1 = r-1, then G_0 is entirely computed by (11).

Since computing coefficient g_{2i+1} needs i products and i additions in $\mathbb{F}_2$, then computing those coefficients from i = 0 up to i = r/2-1 needs

$$(r-2)r/8 \tag{14}$$

products and as many additions.

It is actually worthwile to first compute G_0 since as shown by the following Theorem, solving system (13) needs much less operations than solving system

(12). This corresponds to a situation already exploited by E.R. BERLEKAMP [1] by other means.

Theorem 8.6

Let G be a rational formal power series of $\mathbb{F}_2[[X]]$ without constant term. It only has odd degrees terms iff

(15) $\quad 1+G = (V_1+V_2)/V_2$

where V_1 and V_2 are in $\mathbb{F}_2[X]$, V_1 has only odd degrees terms, V_2 has only even degrees terms and gcd $(V_1,V_2) = 1$.

The condition is necessary

By hypothesis, there exist polynomials U and V from $\mathbb{F}_2[X]$; U(0) = V(0) = 1, gcd (U,V) = 1 such that

(16) $\quad U(1+G) = V.$

We have to prove that $U = V_2$, with $\bar{V}_2 = V_2$ and $U + V = V_1$, with $\tilde{V}_1 = V_1$.

We write $U = \tilde{U} + \bar{U}$. Then $\bar{U}\,G$ (resp. $\tilde{U}\,G$) is a formal power series with only odd degrees terms (resp. even degrees terms). Since $\tilde{U}\,G + \bar{U}\,G$ is a polynomial, and since U(0) = 1, then $\bar{U}\,G$ is necessarily a non-zero polynomial V_1 such that $\tilde{V}_1 = V_1$. By putting $V_2 = \bar{U}$ we obtain (15). We may assume that gcd $(U_2, V_1) = 1$. Indeed V_1 writes $X\,V_0^2$ and V_2 writes T^2. Thus the greatest commun factor to V_1 and V_2 is a square, that is a polynomial with only even degrees terms. By writing $Y=X^2$, it is seen that dividing both members of the relation.

(17) $\quad T^2(X^{-1}G) = V_0^2$

by that polynomial leads to a relation

(18) $\quad V_2^* G = V_1^*,$

where $\bar{V}_2^* = V_2^*$ and $\tilde{V}_1^* = V_1^*$.

The condition is sufficient

If $G = V_1 V_2^{-1}$, since $\tilde{V}_1 = V_1$ and $\bar{V}_2 = V_2$, then V_2^{-1} has only even degrees terms and consequently $G = \tilde{G}$.

Corollary

Let S be a rational formal power series. When writing $S = \tilde{S} + \bar{S}$, we have that $\bar{S} = S^2$ iff

$$1 + S = V_2/(V_1+V_2),$$

where $\bar{V}_2 = V_2$ and $\tilde{V}_1 = V_1$.

8.7 How the decoding algorithm simplifies

Knowing G, we have to compute V_1 and V_2 with the properties of Theorem 8.6 such that $V_2 G = V_1$. We write

(19) $\quad G = X^{2k+1} G_0,$

where G_0 has a non zero constant term, and we necessarily have that $V_1 = X^{2k+1} V_0$ and that $V_2(0) = V_0(0) = 1$.

We thus write $G_0 = H_0^2$, $V_2 = W_2^2$ and $V_0 = W_0^2$. Now knowing H_0, we have to find W_2 and W_0 such that

(20) $\quad W_2 H_0 = W_0.$

It is thus seen that solving (13) reduces to solving (20) and the degrees of all polynomials involved in the process of the iterative Euclidean algorithm are reduced by half.

9 AN EXAMPLE UNDER MACSYMA

We here give command lines under MACSYMA which operate over the field Q of rationals the Normalized Extended Euclidean Algorithm.

We start with polynomials p and q and we first compute the polynomial called "syndrom" given on line d17 which has the property that

$$q.\text{syndrom} = p \bmod X^n,$$

where $n = 2 \deg q+1$.

Now, given polynomial "syndrom", the sequence of iterations stops as soon as $\deg R_{i+1}$, in the listing which is deg [i+1], is smaller than d[j+1]. As it can be seen in (C26), this means, with the notation in 5.2, that the newly computed $d_{ji+1} = d_{j+1}$ verifies $d_{ji+1} \geq n/2+1$.In the example dealt with, we see in line (e72) that deg [i+1] = $\deg R_{i+1} = n - d_{ji+1} = 3$ (Property 4.1), which is verified in line (d75).

On the other hand, we have by Property 6.2 that

$$G_{ji} = G' t_{i+1} = X^{d_{ji+1}} U_{i+1}$$

Since G_{ji} is the G[j] of the listing, command line (c74) is justified.

```
(c3) "We start with two polynomials p and q. We first compute the first n
terms of the expansion of the fraction p/q "$

(c4) p:1+x+x↑3;
                                   3
(d4)                              x  + x + 1

(c5) q:1+x+x↑4;
                                   4
(d5)                              x  + x + 1

(c6) showtime:true;
Time= 3 msec.
(d6)                                 true

(c7) n:2*hipow(q,x)+1;
Time= 10 msec.
(d7)                                   9

(c8) d0:hipow(p,x);
Time= 6 msec.
(d8)                                   3

(c9) d1:hipow(q,x);
Time= 6 msec.
(d9)                                   4

(c10) pp:subst(1/x,x,p);
Time= 14 msec.
                                   1    1
(d10)                              - + ---- + 1
                                   x      3
                                          x
```

```
(c11) qq:subst(1/x,x,q);
Time= 12 msec.
                                    1   1
(d11)                               - + -- + 1
                                    x    4
                                         x

(c12) reciprocal_p:expand(pp*x^d0);
Time= 44 msec.
                                     3    2
(d12)                               x  + x  + 1

(c13) reciprocal_q:expand(qq*x^d1);
Time= 22 msec.
                                     4    3
(d13)                               x  + x  + 1

(c14) p1:expand(reciprocal_p*x^(d1-d0+n-1));
Time= 30 msec.
                                   12    11    9
(d14)                             x   + x   + x

(c15) divide(p1,reciprocal_q,x);
Time= 102 msec.
          8    5      4      3      2                3      2
(d15)    [x  + x  - 2 x  + 2 x  - 2 x  + x + 1, - 3 x  + 2 x  - x - 1]

(c16) reciprocal_syndrom:first(%);
Time= 6 msec.
                     8    5      4      3      2
(d16)               x  + x  - 2 x  + 2 x  - 2 x  + x + 1
```

```
(c17) syndrom:rat(subst(1/x,x,reciprocal_syndrom)*x^(n-1));
Time= 150 msec.
                      8    7      6      5      4    3
(d17)/R/             x  + x  - 2 x  + 2 x  - 2 x  + x  + 1

(c18) "This syndrom is now considered the data to our problem. The iterative
process given in Appendix 4 will then be performed "$
Time= 1 msec.

(c19) G[0]:x;
Time= 7 msec.
(d19)                                  x

(c20) Gprime[0]:0;
Time= 7 msec.
(d20)                                  0

(c21) d[0]:0;
Time= 6 msec.
(d21)                                  0

(c22) delta:1;
Time= 3 msec.
(d22)                                  1

(c23) j:0;
Time= 3 msec.
(d23)                                  0

(c24) deg[-1]:n;
Time= 7 msec.
(d24)                                  9
```

```
(c25) i:0;
Time= 3 msec.
(d25)                                  0

(c26) block(
          debut,

                    discrepancy[j]:((-1)↑i)*coeff(G[j]*syndrom,x,n),
                     ldisplay(discrepancy[j]),
                     if discrepancy[j]=0 then go(enfin),
                     if j<2*d[j] then go(suite),
                     ldisplay(i),
                     ldisplay(d[j]),
                     d[j+1]:j+1-d[j],
                     deg[i]:n-d[j+1],
                     ldisplay(deg[i]),
                     ldisplay(d[j+1]),
                     U[i]:rat(G[j]/x↑d[j+1]),
                     ldisplay(U[i]),
                     if deg[i]<d[j+1] then go(fin),
                     gregistre:G[j],
                     G[j+1]:
                     G[j]+
                     (discrepancy[j]*Gprime[j])/delta,
                     G[j+1]:rat(x*G[j+1]),
                     Gprime[j+1]:gregistre,
                     delta:discrepancy[j],
                     i:i+1,
                     j:j+1,
                     ldisplay(Gprime[j]),
                     ldisplay(G[j]),
                     go(debut),
          suite,
```

```
                    G[j+1]:
                    G[j]+(discrepancy[j]*Gprime[j])/delta,
                    G[j+1]:rat(x*G[j+1]),
                    Gprime[j+1]:Gprime[j],
                    d[j+1]:d[j],
                    j:j+1,
                    ldisplay(Gprime[j]),
                    ldisplay(G[j]),
                    go(debut),
          enfin,
                    G[j+1]:rat(x*G[j]),
                    Gprime[j+1]:Gprime[j],
                    d[j+1]:d[j],
                    j:j+1,
                    ldisplay(Gprime[j]),
                    ldisplay(G[j]),
                    go(debut),
            fin
                    )$
```

(e26)/R/
$$\text{discrepancy}_0 = 1$$

(e27)
$$i = 0$$

(e28)
$$d_0 = 0$$

(e29)
$$\text{deg} = 8$$

(e30)
$$d_1 = 1$$

(e31)/R/
$$u_0 = 1$$

(e32)
$$gprime = x_1$$

(e33)/R/
$$g_1 = x^2$$

(e34)/R/
$$discrepancy = -1$$

(e35)
$$gprime = x_2$$

(e36)/R/
$$g_2 = x^3 - x^2$$

(e37)/R/
$$discrepancy = 3$$

(e38)
$$i = 1$$

(e39)
$$d_2 = 1$$

(e40)
$$deg = 7$$

(e41)
$$d_3 = 2$$

(e42)/R/
$$u_1 = x - 1$$

(e43)/R/
$$gprime_3 = x^3 - x^2$$

(e44)/R/
$$g_3 = x^4 - x^3 + 3x^2$$

(e45)/R/
$$discrepancy_3 = 7$$

(e46)/R/
$$gprime_4 = x^3 - x^2$$

(e47)/R/
$$g_4 = \frac{3x^5 + 4x^4 + 2x^3}{3}$$

(e48)/R/
$$discrepancy_4 = -\frac{2}{3}$$

(e49)
$$i = 2$$

(e50)
$$d_4 = 2$$

(e51) $$\deg_2 = 6$$

(e52) $$d_5 = 3$$

(e53)/R/ $$u_2 = \frac{3x^2 + 4x + 2}{3}$$

(e54)/R/ $$gprime_5 = \frac{3x^5 + 4x^4 + 2x^3}{3}$$

(e55)/R/ $$g_5 = \frac{9x^6 + 12x^5 + 4x^4 + 2x^3}{9}$$

(e56)/R/ $$\text{discrepancy}_5 = \frac{11}{9}$$

(e57)/R/ $$gprime_6 = \frac{3x^5 + 4x^4 + 2x^3}{3}$$

(e58)/R/ $$g_6 = \frac{2x^7 - x^6 - 4x^5 - 2x^4}{2}$$

(e59)/R/ $$\text{discrepancy}_6 = -\frac{3}{2}$$

(e60) $$i = 3$$

(e61) $$d_6 = 3$$

(e62) $$\deg_3 = 5$$

(e63) $$d_7 = 4$$

(e64)/R/ $$u_3 = \frac{2x^3 - x^2 - 4x - 2}{2}$$

(e65)/R/ $$\text{gprime}_7 = \frac{2x^7 - x^6 - 4x^5 - 2x^4}{2}$$

(e66)/R/ $$g_7 = \frac{4x^8 - 2x^7 + x^6 + 8x^5 + 6x^4}{4}$$

(e67)/R/ $$\text{discrepancy}_7 = -\frac{3}{4}$$

(e68)/R/
$$gprime_{8} = \frac{2x^{7} - x^{6} - 4x^{5} - 2x^{4}}{2}$$

(e69)/R/
$$g_{8} = x^{9} + x^{6} + x^{5}$$

(e70)/R/
$$discrepancy_{8} = 0$$

(e71)/R/
$$gprime_{9} = \frac{2x^{7} - x^{6} - 4x^{5} - 2x^{4}}{2}$$

(e72)/R/
$$g_{9} = x^{10} + x^{7} + x^{6}$$

(e73)/R/
$$discrepancy_{9} = 1$$

(e74)
$$i = 4$$

(e75)
$$d_{9} = 4$$

(e76)
$$deg_{4} = 3$$

```
(e77)                                   d    = 6
                                         10

                                          4
(e78)/R/                           u  = x  + x + 1
                                    4

Time= 1882 msec.

(c79) denominateur:rat(G[j]/x^d[j+1]);
Time= 27 msec.
                                          4
(d79)/R/                                 x  + x + 1

(c80) numerateur:remainder(syndrom*denominateur,x^n)$
Time= 26 msec.

(c81) display(R[i]:numerateur);
                                           3
                                   r  = x  + x + 1
                                    4

Time= 9 msec.
(d81)                                    done

Time= 9801 msec.
(d82)                                BATCH DONE

(c83) closefile(iterneuc);
```

APPENDIX 1

The iterative process for the usual Euclidean algorithm.

The degree of R_{-1} is denoted by n. We have that deg $R_0 < n$.

1 Initialization

$j \leftarrow 0$, $E_0 \leftarrow X\ R_0$, $E'_0 \leftarrow R_{-1}$, $d_0 \leftarrow 0$, $\Delta \leftarrow [X^n]R_{-1}$.

2 Picking out the discrepancy Δ_j

$\Delta_j \leftarrow [X^n]E_j$.

3a If $\Delta_j = 0$, then shift E_j :

$d_{j+1} \leftarrow d_j$, $E_{j+1} \leftarrow X\ E_j$, $E'_{j+1} \leftarrow E'_j$, $\Delta \leftarrow \Delta$.

3b If $\Delta_j \neq 0$ and $2d_j \leq j$, then interchange E_j and E'_j, interchange Δ_j and Δ, finally modify d_j

$M \leftarrow E_j$, $E_j \leftarrow E'_j$, $E'_j \leftarrow M$,
$T \leftarrow \Delta_j$, $\Delta_j \leftarrow \Delta$, $\Delta \leftarrow T$,
$E_{j+1} \leftarrow E_j - \Delta_j\ \Delta^{-1}E'_j$,
$E_{j+1} \leftarrow X\ E_{j+1}$, $E'_{j+1} \leftarrow E'_j$,
$d_{j+1} \leftarrow j+1 - d_j$.

3c If $\Delta_j \neq 0$ and $2d_j > j$, then d_j, Δ_j and Δ remains unchanged.

$E_{j+1} \leftarrow E_j - \Delta_j\ \Delta^{-1}E'_j$,
$E_{j+1} \leftarrow X\ E_{j+1}$, $E'_{j+1} \leftarrow E'_j$,
$d_{j+1} \leftarrow d_j$, $\Delta \leftarrow \Delta$.

4 The loop closes up

$j \leftarrow j+1$, goto 2.

APPENDIX 2

Iterative process for computing polynomials U_i and V_i in the extended Euclidean algorithm without actually computing the sequence R_i of remainders

The values to start with are R_{-1} and R_0 as in Appendix 1, and $U_{-1} = V_0 = 0$, $V_{-1} = U_0 = 1$.

1 Initialization

$$j \leftarrow 0,\quad G_0 \leftarrow X\,U_0,\quad G'_0 \leftarrow U_{-1},\quad d_0 \leftarrow 0,\quad \Delta \leftarrow [X^n]R_{-1},$$

$$i \leftarrow 0,\quad H_0 \leftarrow X\,V_0,\quad H'_0 \leftarrow V_{-1}.$$

2 Picking out the discrepancy Δ_j

$$\Delta_j \leftarrow (-1)^i([X^n]G_j\,R_0 - [X^n]\,H_j\,R_{-1})$$

3a If $\Delta_j = 0$, then shift G_j, H_j

$$d_{j+1} \leftarrow d_j,\quad G_{j+1} \leftarrow X\,G_j,\quad G'_{j+1} \leftarrow G'_j,\quad \Delta \leftarrow \Delta,$$
$$H_{j+1} \leftarrow X\,H_j,\quad H'_{j+1} \leftarrow H'_j.$$

3b If $\Delta_j \neq 0$ and $2d_j \leq j$, then interchange G_j and G'_j, H'_j and H'_j, Δ_j and Δ, respectively. Modify G_j and H_j and finally modify d_j

$$M \leftarrow G_j,\quad G_j \leftarrow G'_j,\quad G'_j \leftarrow M,$$
$$N \leftarrow H_j,\quad H_j \leftarrow H'_j,\quad H'_j \leftarrow N,$$
$$T \leftarrow \Delta_j,\quad \Delta_j \leftarrow \Delta,\quad \Delta \leftarrow T,$$
$$G_{j+1} \leftarrow X\,G_j + \Delta_j\,\Delta^{-1}G'_j,$$
$$H_{j+1} \leftarrow H_j + \Delta_j\,\Delta^{-1}H'_j,$$
$$d_{j+1} \leftarrow j+1 - d_j,$$
$$i \leftarrow i+1.$$

3c **If $\Delta_j \neq 0$ and $2d_j > j$, then d_j, Δ_j and Δ remain unchanged**

$G_{j+1} \leftarrow G_j + \Delta_j \Delta^{-1} G'_j$,
$H_{j+1} \leftarrow H_j + \Delta_j \Delta^{-1} H'_j$,
$G_{j+1} \leftarrow X\ G_{j+1}$, $G'_{j+1} \leftarrow G'_j$,
$H_{j+1} \leftarrow X\ H_{j+1}$, $H'_{j+1} \leftarrow H'_j$,
$d_{j+1} \leftarrow d_j, \Delta \leftarrow \Delta$.

4 **The loop closes up**

$j \leftarrow j+1$, goto 2.

APPENDIX 3

Iterative process for the normalized Euclidean algorithm.

The values to start with are R^*_{-1} and R^*_0 with $n = \deg R^*_{-1} > \deg R^*_0$.

1 Initialization

$j \leftarrow 0,\ E_0 \leftarrow X\ R^*_0,\ E'_0 \leftarrow R^*_{-1},\ d_0 \leftarrow 0,\ \Delta \leftarrow [X^n]R^*_{-1}.$

2 Picking out the discrepancy Δ_j

$\Delta_j \leftarrow [X^n]E_j\ .$

3a If $\Delta_j = 0$, then shift E_j

$d_{j+1} \leftarrow d_j,\ E_{j+1} \leftarrow X\ E_j,\ E'_{j+1} \leftarrow E'_j,\ \Delta \leftarrow \Delta.$

3b If $\Delta_j \neq 0$ and $2d_j \leq j$, then compute a new E_j, a new d_j and a new Δ

$E_{j+1} \leftarrow E_j - \Delta_j\ \Delta^{-1}E'_j, E'_{j+1} \leftarrow E_j,$
$E_{j+1} \leftarrow X\,E_{j+1}, \Delta \leftarrow \Delta_j\ ,\ d_{j+1} \leftarrow j+1 - d_j.$

3c If $\Delta_j \neq 0$ and $2d_j > j$, then compute a new E_j

$E_{j+1} \leftarrow E_j - \Delta_j\ \Delta^{-1}E'_j, E'_{j+1} \leftarrow E_j,$
$E_{j+1} \leftarrow X\,E_{j+1}, \Delta \leftarrow \Delta_j.$

4 The loop closes up

$j \leftarrow j+1,$ goto 2.

APPENDIX 4

Iterative process for computing polynomials U_i^* in the normalized extended Euclidean algorithm for $R_{-1}^* = X^n$ and deg $R_0^* < r$ without actually computing the sequence R_i^* or normalized remainders.

The values to start with are R_{-1}^* and R_0^* and $U_{-1}^* = 0$, $U_0^* = 1$

1 Initialization

$j \leftarrow 0$, $G_0 \leftarrow X$, $G'_0 \leftarrow 0$, $d_0 \leftarrow 0$, $\Delta \leftarrow 1$, $i \leftarrow 0$.

2 Picking out the discrepancy Δ_j

$\Delta_j \leftarrow (-1)^i [X^n] G_j R_0^*$.

3a If $\Delta_j = 0$, then shift G_j

$d_{j+1} \leftarrow d_j$, $G_{j+1} \leftarrow X G_j$, $G'_{j+1} \leftarrow G'_j$, $\Delta \leftarrow \Delta$.

3b If $\Delta_j \neq 0$ and $2d_j \leq j$, then compute a new G_j, a new d_j and a new Δ

$G_{j+1} \leftarrow G_j - \Delta_j \Delta^{-1} G'_j$, $G'_{j+1} \leftarrow G_j$,
$G_{j+1} \leftarrow X G_{j+1}$, $\Delta \leftarrow \Delta_j$, $d_{j+1} \leftarrow j+1 - d_j$, $i \leftarrow i+1$.

3c If $\Delta_j \neq 0$ and $2d_j > j$, then compute a new G_j

$G_{j+1} \leftarrow G_j + \Delta_j \Delta^{-1} G'_j$, $G'_{j+1} \leftarrow G'_j$,
$G_{j+1} \leftarrow X G_{j+1}$, $\Delta \leftarrow \Delta_j$.

4 The loop closes up

$j \leftarrow j+1$, goto 2.

REFERENCES

[1] E.R. Berlekamp, "Algebraic Coding Theory" Mc Graw-Hill 1968.

[2] N. Bourbaki, "Eléments de Mathématique" Livre II Algèbre - Chapitre 4 Polynômes et fractions rationnelles. Herman 1959.

[3] JL. Dornstetter, "On the Equivalence Between Berlekamp's and Euclid's Algorithms" IEEE Transactions on Information Theory IT 33 n° 3 May 1987.

[4] F.J. MacWilliams and N.J.A. Sloane "The Theory of Error-Correcting Codes" Amsterdam North Holland 1977.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique